For this todo app, users should ideally interact with only a single page. This main page will contain all the functionality to add, view, edit, and delete todo items, and search for all items with a tag.

In this repository, I've also included the source code to a basic working version of the app (excluding `config/database.yml`). It isn't fully functional nor polished yet, but supports basic CRUD operations for todo items.

Execution Details

Todo items will be represented by a model with these properties:

- *Content:* string
- *Complete*: boolean, default is false
- *Priority*: boolean, default is false
- *Tags*: array of strings, default is an empty array

The main page should have the following components, in order from top to bottom:

- *Form (to add a new todo item)*: Has a single row with a text field for content, and a submit button.
- *Search bar (to filter for tags)*: Upon searching for a specific tag, the table below will refresh to display only todo items with the tag.
- *Table (to display todo items)*: Items will be displayed in a table, one item per row. Each item can be completed by checking a box, given priority by clicking a star, and edited/deleted along with its tags.

Basic Use Cases

In the normal flow of events, users add a todo item using a form; the item is then displayed in the table below. Each todo item is displayed in the table as a row, which contains a checkbox, a star, its content, a pencil icon, and a trash icon (from left to right). From there, users can check off a completed item, give the item priority, edit or delete the item, and add or delete its tags.

Here's how the actions are performed, in greater detail:

Add a todo item
Use the form on top of the main page to add a new todo item. Only the *content* of the item needs to specified; other properties (i.e. *complete*, *priority*, *tags*) can be added or modified later.

Check or change the priority of a todo item
To check a todo item, click on its checkbox, which toggles its *complete* property. Checked items will be sent to the bottom and appear striked out.

To give a todo item priority, click on the star, which toggles its *priority* property. Items with priority will be shifted to the top.

Edit or delete a todo item
To edit a todo item, click on the pencil icon. A modal pops up, containing a form with a text field to update its *content*. To delete a todo item, click on the trash icon.

Add or delete the tag of a todo item
A todo item's *tags* will be displayed below its content. A small form beside the todo item will be present for adding a new tag. All existing tags will also have a cross icon beside it to delete it.

More Details

Bootstrap will be used as the front-end framework. I've chosen it because it provides modals (which can be used to edit todo items), and also glyphicons, which look clean and are handy to use in lieu of obtrusive words like 'Edit' or 'Delete'.

PostgreSQL will be used for the database. While I haven't tried hosting this app on Heroku yet, it seems to require Postgres, so I've decided to forgo SQLite in favour of Postgres from the get-go to familiarize myself with it, and spare myself some trouble in switching databases later on.

Problems

I'm not familiar with Postgres. In general, I'm not very skilled when it comes to relational databases or writing SQL queries, which points to a gap in my understanding that I've yet to close fully. Setting up Postgres and using `psql` so far has also been a pain; several times my database mysteriously crashed while throwing errors I couldn't resolve, forcing me to redo the project. But after looking at more online docs about Postgres and Rails, my app seems to work fine this time.

I'm not too sure how to best implement the tagging system, either. Displaying all of a todo item's tags together with its content might clutter the todo list, but hiding the tags or having a separate pop-up for them might risk them being invisible to the user. I'll figure this one out as I go along.

AJAX sounds like an attractive option, because it allows for the main page to be uploaded without constant reloading. Considering that I'm intending to put all of the functionality in one page, using AJAX should help to smooth things out for a better user experience. I haven't looked into it yet, but I'll try to learn it along the way and integrate it into the app.